

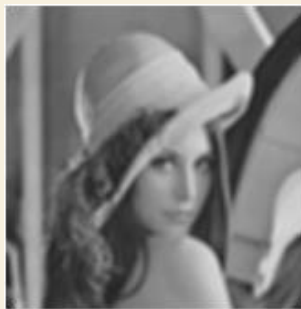
ЭЛЕМЕНТЫ ТЕХНИЧЕСКОГО ЗРЕНИЯ

ФИЛЬТРАЦИЯ В
ЧАСТОТНОЙ
ОБЛАСТИ

ИЗОБРАЖЕНИЕ: ЧТО МОЖЕТ ПОЛУЧИТЬСЯ ПЛОХО?



Impulse noise



ПРЕОБРАЗОВАНИЯ ФУРЬЕ

- Преобразование Фурье – интегральное преобразование вида

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx, \quad f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(\omega) e^{i\omega x} d\omega$$

- Разложение сигнала на частоты и амплитуды, то есть переход от временного пространства в частотное
- Для многомерного случая

$$\hat{f}(\vec{\omega}) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}^n} f(\vec{r}) e^{-i\vec{\omega}\vec{r}} d\vec{r}, \quad f(\vec{x}) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}^n} f(\vec{\omega}) e^{i\vec{\omega}\vec{r}} d\vec{\omega}$$

ДИСКРЕТНОЕ ПРЕОБРАЗОВАНИЕ

- Аналогично непрерывному, но вместо интеграла – сумма

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn}, \quad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn}$$

- Если для непрерывного преобразования частоты непрерывны, а их количество бесконечно, здесь их всего N
- Обычно используется быстрое преобразование Фурье (FFT)
- Количество вычислений порядка $O(n \cdot \log(n))$

ДЛЯ ЧЕГО?

- В пространстве Фурье свёртка превращается в умножение-быстрее считаем
- Переход в частотную область – фильтры высоких и низких частот
- Фильтр низких частот – те фильтры, что мы уже рассматривали

НЕРАВНОМЕРНАЯ ПОДСВЕТКА

- Предположим, что у нас есть система из камеры и подсветки
- Подсветка приводит к неравномерной освещённости кадра
- В движении освещённость меняется
- Как этот эффект устранить?



ФИЛЬТР ВЫСОКИХ ЧАСТОТ

- Мы можем попробовать представить изображение как комбинацию распределения освещённости от подсветки и «собственной» яркости
- При этом можно рассматривать освещённость как гармоники с низкой частотой
- К сожалению, комбинация не аддитивная, а мультипликативная
- Но превращается в аддитивную при логарифмировании!

```

#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>

using namespace cv;
using namespace std;

void CalcFFT(Mat input, Mat &complex);
void CalcIFFT(Mat complex, Mat &output);
void fftshift(const Mat& inputImg, Mat& outputImg);
Mat MakeHPF(Size size, int D, int n, float high_h_v_TB, float low_h_v_TB);

int main()
{
    Mat imgIn = imread("Picture1.png", IMREAD_COLOR);
    if (imgIn.empty()) //check whether the image is loaded or not
    {
        cout << "ERROR : Image cannot be loaded..!!" << endl;
        return -1;
    }
    cvtColor(imgIn, imgIn, CV_BGR2GRAY);
    imshow("Original", imgIn);

    int high_h_v_TB = 120;
    int low_h_v_TB = 80;
    int D = 10; // radius of band pass filter parameter
    int order = 2;

    Mat imgMagn, imgPhase, imgOut, imgComplex;
    CalcFFT(imgIn, imgComplex);
    fftshift(imgComplex, imgComplex);
    Mat filter = MakeHPF(imgComplex.size(), D, order, high_h_v_TB, low_h_v_TB);
    mulSpectrums(imgComplex, filter, imgComplex, 0);
    fftshift(imgComplex, imgComplex);
    CalcIFFT(imgComplex, imgOut);

    Rect roi(0,0, imgIn.cols-10, imgIn.rows-10);
    imgOut = imgOut(roi);
    normalize(imgOut, imgOut, 0, 1, NORM_MINMAX);

    imshow("After", imgOut);

    waitKey(0);
    return 0;
}

```


ПРЯМОЕ И ОБРАТНОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ

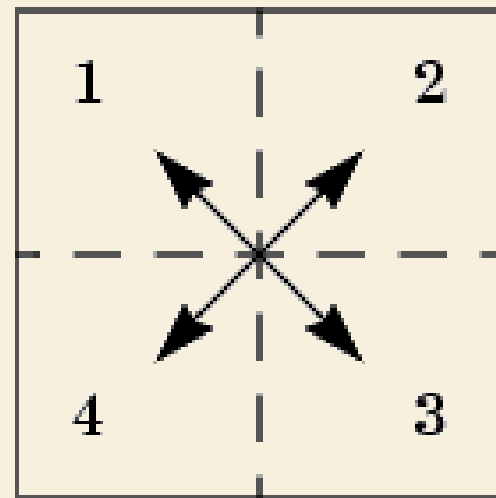
```
void CalcFFT(Mat input, Mat &complex)
{
    int padRows = getOptimalDFTSize(input.rows);
    int padCols = getOptimalDFTSize(input.cols);
    // Преобразование Фурье эффективно считается не для всех размеров
    Mat imgIn, padded, imgComplex;
    input.convertTo(imgIn, CV_32F);
    imgIn += 1; // Чтобы мои значения были больше 1
    log(imgIn, imgIn); // логарифмирование - вместо умножения получили сложение
    copyMakeBorder(imgIn, padded, 0, padRows - imgIn.rows, 0, padCols - imgIn.cols, BORDER_CONSTANT);

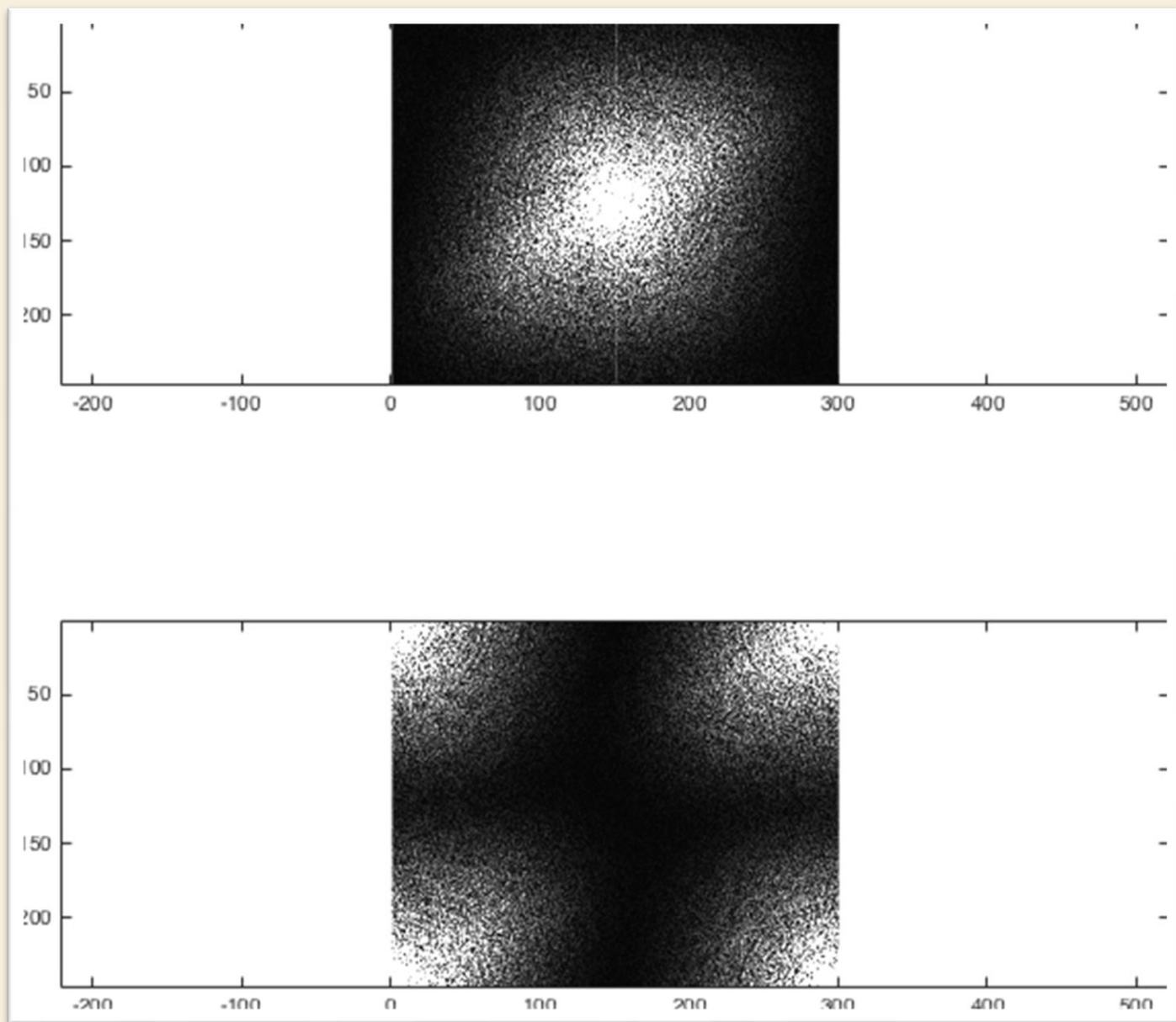
    Mat planes [] = {Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F)};
    merge(planes, 2, imgComplex);
    dft(imgComplex, complex, DFT_SCALE);
}

void CalcIFFT(Mat complex, Mat &output)
{
    dft(complex, output, DFT_INVERSE | DFT_REAL_OUTPUT);
    exp(output, output); // мы до этого в логарифмическом масштабе работали
}
```

FFT_SHIFT – ЗАЧЕМ?

- Получающееся изображение в не очень удобном для нас виде – ноль в нижнем левом углу
- А мы привыкли, чтобы ноль был в центре
- Поэтому чуть-чуть меняем изображение





```
void fftshift(const Mat& inputImg, Mat& outputImg)
{
    outputImg = inputImg.clone();
    int cx = outputImg.cols / 2;
    int cy = outputImg.rows / 2;
    Mat q0(outputImg, Rect(0, 0, cx, cy));
    Mat q1(outputImg, Rect(cx, 0, cx, cy));
    Mat q2(outputImg, Rect(0, cy, cx, cy));
    Mat q3(outputImg, Rect(cx, cy, cx, cy));
    Mat tmp;
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
}
```

В КОДЕ

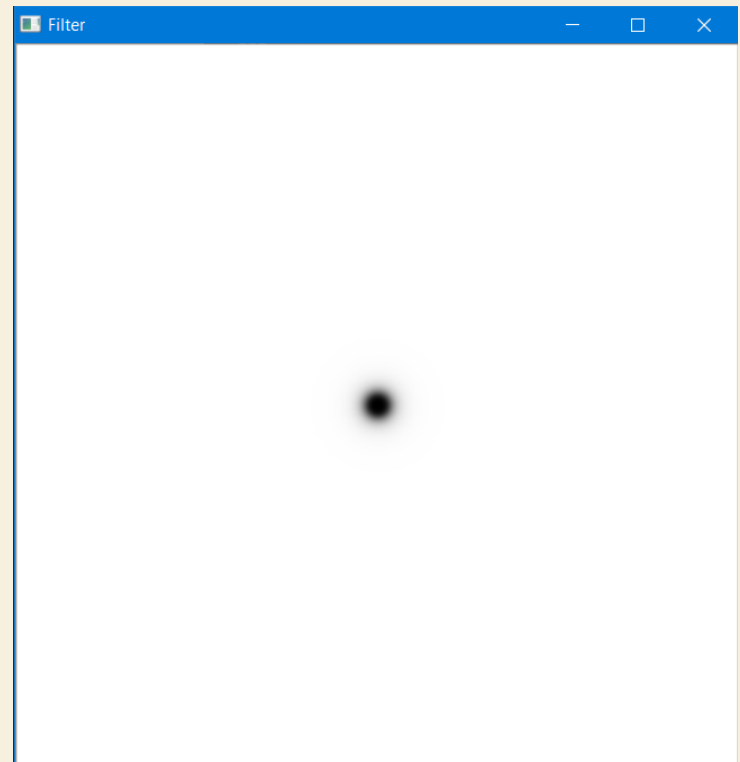
ФИЛЬТР

```
Mat MakeHPF(Size size, int D, int n, float high_h_v_TB, float low_h_v_TB)
{
    Mat filter(size, CV_32F);
    Point centre = Point(filter.rows/2, filter.cols/2);
    double radius;
    float upper = (high_h_v_TB * 0.01);
    float lower = (low_h_v_TB * 0.01);

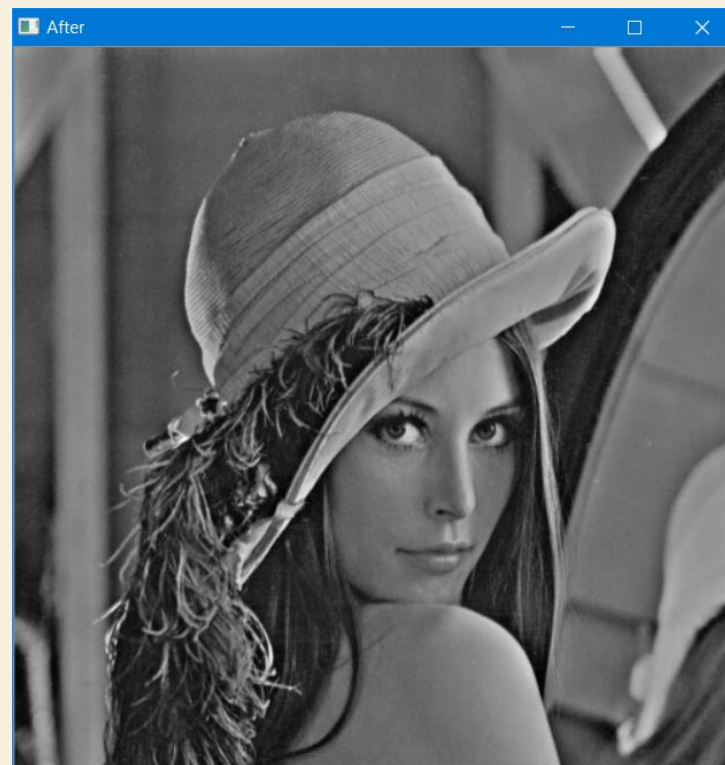
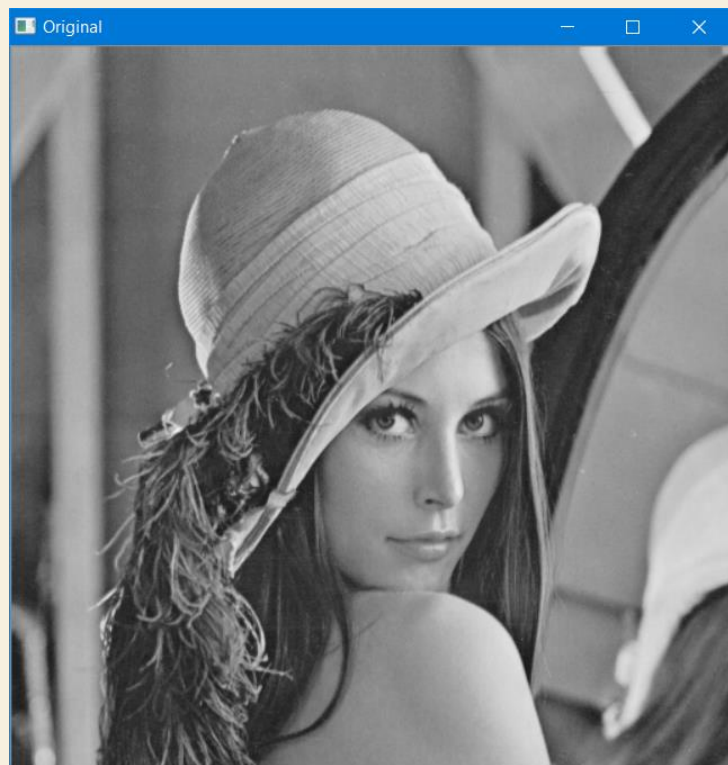
    for(int i = 0; i < filter.rows; i++)
    {
        for(int j = 0; j < filter.cols; j++)
        {
            radius = (double) sqrt(pow((i - centre.x), 2.0) + pow((double) (j - centre.y), 2.0));
            filter.at<float>(i,j) =\
                ((upper - lower) * (1/(1 + pow((double) (D/radius), (double) (2*n)))))) + lower;
        }
    }
    Mat planes[2] = { Mat_<float>(filter.clone()), Mat::zeros(filter.size(), CV_32F) };
    merge(planes,2,filter);
    normalize(planes[0], planes[0], 0, 1, NORM_MINMAX);
    imshow("Filter", planes[0]);
    return filter;
}
```

КАК ОН ВЫГЛЯДИТ

- Представление в частотной области
- В центре у нас значение нижнего порога
- При перемножении спектров фильтр обрезает низкие частоты (сосредоточены в центре)



ЧТО ПОЛУЧАЕТСЯ



И ЕЩЁ



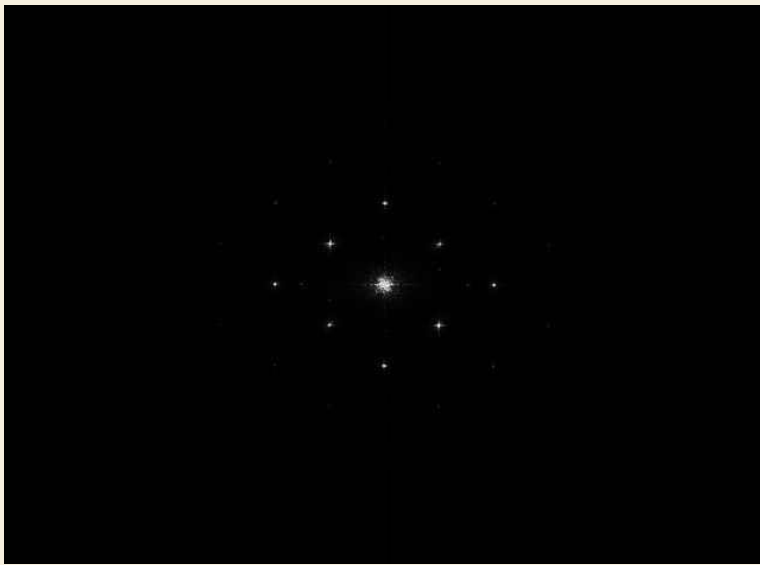
ФИЛЬТР ПЕРИОДИЧЕСКИХ ПОМЕХ

- Идея та же самая
- Периодическая помеха — помеха с определённой частотой
- Значит она будет локализована в пространстве Фурье

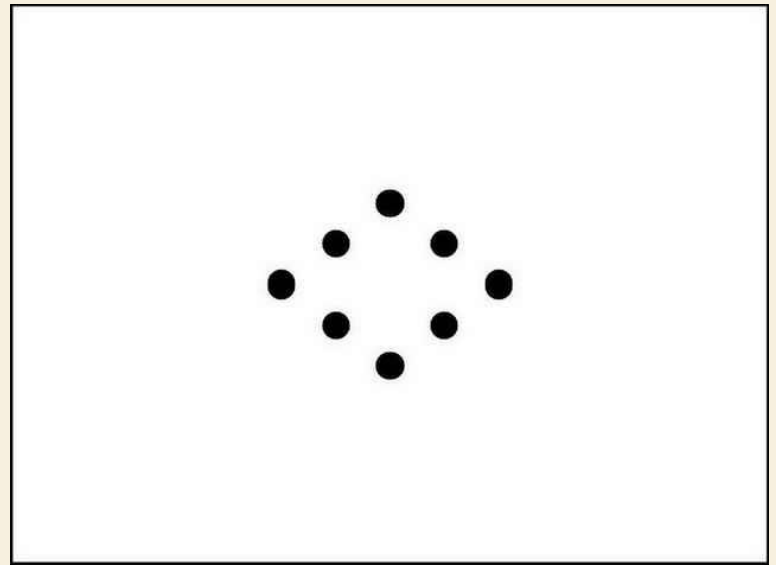


В ЧАСТОТНОЙ ОБЛАСТИ

ИЗОБРАЖЕНИЕ



ФИЛЬТР



СОЗДАНИЕ ФИЛЬТРА...

```
void synthesizeFilterH(Mat& inputOutput_H, Point center, int radius)
{
    Point c2 = center, c3 = center, c4 = center;
    c2.y = inputOutput_H.rows - center.y;
    c3.x = inputOutput_H.cols - center.x;
    c4 = Point(c3.x, c2.y);
    circle(inputOutput_H, center, radius, 0, -1, 8);
    circle(inputOutput_H, c2, radius, 0, -1, 8);
    circle(inputOutput_H, c3, radius, 0, -1, 8);
    circle(inputOutput_H, c4, radius, 0, -1, 8);
}
```

...И ЕГО ПРИМЕНЕНИЕ

```
Rect roi = Rect(0, 0, imgIn.cols & -2, imgIn.rows & -2);  
Mat H = Mat(roi.size(), CV_32F, Scalar(1));  
const int r = 21;  
synthesizeFilterH(H, Point(705, 458), r);  
synthesizeFilterH(H, Point(850, 391), r);  
synthesizeFilterH(H, Point(993, 325), r);
```

РЕЗУЛЬТАТ



УСТРАНЕНИЕ РАЗМЫТИЯ

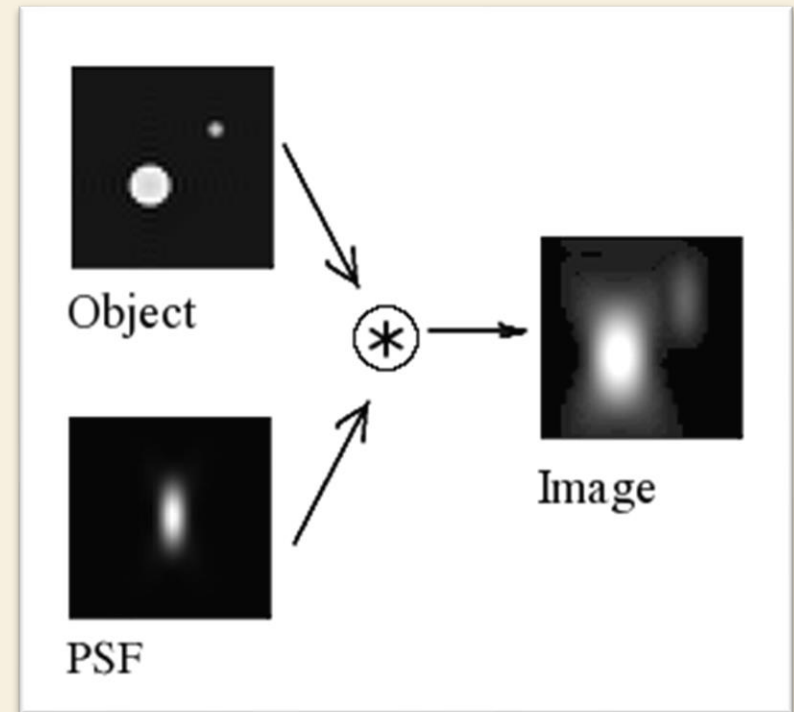
- Мы снова попробуем применять преобразование Фурье
- Для этого нам нужна модель размытия
- В частотном диапазоне:

$$S = H \cdot U + N$$

- Здесь
 - S – результирующее изображение
 - U – исходное изображение
 - N – аддитивный шум
 - H – функция рассеяния точки

ФУНКЦИЯ РАССЕЯНИЯ ТОЧКИ

- Point spread function (PSF)
- Показывает, во что превращается точка при размытии
- Зависит от условий съёмки



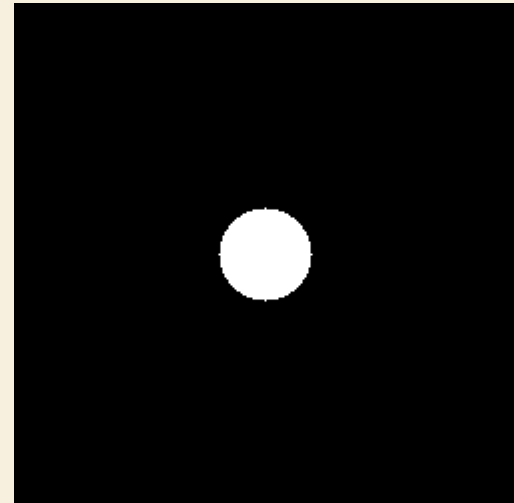
ЧАСТНЫЙ СЛУЧАЙ

- Если мы знаем PSF, то можем найти изображение

$$U = H_{\omega} S$$

- Примем простейшую PSF в виде круга
- Тогда фильтр можно составить в виде

$$H_{\omega} = \frac{H}{H^2 + \frac{1}{SNR}}$$



ПРИМЕНЕНИЕ

```
Rect roi = Rect(0, 0, imgIn.cols & -2, imgIn.rows & -2);  
//Hw calculation (start)  
Mat Hw, h;  
calcPSF(h, roi.size(), R);  
calcWnrFilter(h, Hw, 1.0 / double(snr));  
//Hw calculation (stop)  
filter2DFreq(imgIn(roi), imgOut, Hw);
```

PSF

- Просто рисуем кружок

```
void calcPSF(Mat& outputImg, Size filterSize, int R)
{
    Mat h(filterSize, CV_32F, Scalar(0));
    Point point(filterSize.width / 2, filterSize.height / 2);
    circle(h, point, R, 255, -1, 8);
    Scalar summa = sum(h);
    outputImg = h / summa[0];
}
```

СОЗДАНИЕ ФИЛЬТРА

```
void calcWnrFilter(const Mat& input_h_PSF, Mat& output_G, double nsr)
{
    Mat h_PSF_shifted;
    fftshift(input_h_PSF, h_PSF_shifted);
    Mat planes[2] = { Mat_<float>(h_PSF_shifted.clone()), Mat::zeros(h_PSF_shifted.size(), CV_32F) };
    Mat complexI;
    merge(planes, 2, complexI);
    dft(complexI, complexI);
    split(complexI, planes);
    Mat denom;
    pow(abs(planes[0]), 2, denom);
    denom += nsr;
    divide(planes[0], denom, output_G);
}
```

ФИЛЬТРАЦИЯ

```
void filter2DFreq(const Mat& inputImg, Mat& outputImg, const Mat& H)
{
    Mat planes[2] = { Mat_<float>(inputImg.clone()), Mat::zeros(inputImg.size(), CV_32F) };
    Mat complexI;
    merge(planes, 2, complexI);
    dft(complexI, complexI, DFT_SCALE);

    Mat planesH[2] = { Mat_<float>(H.clone()), Mat::zeros(H.size(), CV_32F) };
    Mat complexH;
    merge(planesH, 2, complexH);
    Mat complexIH;
    mulSpectrums(complexI, complexH, complexIH, 0);

    idft(complexIH, complexIH);
    split(complexIH, planes);
    outputImg = planes[0];
}
```

РЕЗУЛЬТАТ

